Implementation and Evaluation of Time Synchronization Mechanisms for Generic Embedded Systems for Time Sensitive Networking (TSN)

Leon Schürmann
University of Stuttgart
Stuttgart, Germany
st169727@stud.uni-stuttgart.de

Frank Dürr
Institute of Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
frank.duerr@ipvs.uni-stuttgart.de

***Abstract*—Precise time synchronization is essential for many real-time systems, for instance in the field of the Industrial Internet of Things. To this end, the IEEE has defined the Precision Time Protocol (IEEE 1588) to synchronize devices in IEEE 802 networks, including Ethernet and its extensions for real-time communication known by the term Time Sensitive Networking (TSN).**

This paper analyzes the feasibility of implementing IEEE 1588 Precision Time Protocol based time synchronization and syntonization, using commodity microcontrollers without hardware-based assistance for timestamping IEEE 1588 messages. A Device Under Test, built on a LiteX and RISC-V based FPGA and running the Tock embedded OS, is used to implement an IEEE 1588 slave clock. This clock synchronizes to a precise Grandmaster clock. A time-to-digital-converter measures the system's synchronization accuracy while subject to internal and external simulated system conditions. The acquired data is analyzed and used to derive conclusions about the impact of specific system conditions and architectural choices on the achievable synchronization accuracy.

I. Introduction

Time Sensitive Networking (TSN) has a growing number of applications in both industry and consumer facing scenarios. Precise time synchronization between networked devices is a strict requirement in a wide variety of application areas, such as the Industrial Internet of Things (IIoT), distributed audio interfaces or Smart Grids.

TSN describes a set of open and vendor-independent IEEE standards that enable real-time communication and precise time synchronization across standard IEEE 802.3 (Ethernet) networks. In particular, the time synchronization aspect of TSN is implemented using the IEEE 1588 Precision Time Protocol (PTP). Using preexisting network infrastructures and vendor-independent standards, such as Ethernet, for time synchronization between devices can have a significant economical advantage compared to competing proprietary solutions. Furthermore, PTP is specified such that the participating network stations do not necessarily require hardware support for this protocol. On the one hand, stations with hardware support for PTP may achieve a substantially higher degree of time synchronization precision compared to an implementation without explicit hardware support. On the other hand, if a software-only implementation of PTP can deliver the required time synchronization precision for the target application area, time synchronization can be retrofitted in these systems without hardware modifications.

Commonly, application areas of TSN utilize highly-specialized embedded systems centered around microcontroller platforms to implement their application logic, as opposed to conventional personal computing devices. The availability of open TSN standards facilitates hardware-assisted implementations of PTP in such hardware components. In fact, a number of microcontroller platforms offer dedicated hardware assistance for PTP. Still, many existing and deployed hardware platforms do not feature explicit hardware support for PTP. This raises the question: what degree of synchronization precision of an implementation of PTP can be expected, using only standard microcontroller systems without hardware assistance?

In an effort to answer the aforementioned question, this paper shall analyze the feasibility and performance of time synchronization using PTP in embedded systems without hardware support.

Contributions of this paper include an implementation of a subset of the IEEE 1588 Precision Time Protocol for Tock, a novel embedded operating system, as well as a port of Tock to LiteX System on a Chips (SoCs) with a RISC-V processor and an implementation of hardware timestamping infrastructure for LiteEth, a Field Programmable Gate Array (FPGA) Ethernet Media Access Control (MAC) core.

The rest of the paper is structured as follows: first, an overview over the relevant mechanisms of PTP is provided in Section II. Section III outlines this work's problem statement. In Section IV, an analysis of important system properties for a measurement setup is conducted. Based on these results, Section V describes a measurement architecture including hardware and software specifications, as well as measurement approaches. Further, statistical methods to analyze acquired data are elaborated in Section VI. Section VII uses these statistical methods to perform time synchronization measurements and interpret the observed results. Finally, Section VIII concludes this paper and discusses potential future work.

II. Overview of the Precision Time Protocol

The IEEE Standard 1588 defines the Precision Time Protocol (PTP), a protocol for accurate synchronization and syntonization of real-time clocks between devices in networked distributed systems. In particular, PTP can be used to synchronize and syntonize clocks of devices connected using commodity Ethernet connections and can support clocks of different precision, resolution and stability guarantees [1, p. 14].

Clock syntonization describes the process of adjusting the frequency of two independent clocks such that they run at the same rate, hence agree on the length of some fixed time interval. Clock or time synchronization describes clocks agreeing on the current time [2]. Hence, clock syntonization is a measure of the relative agreement of two independently running clocks, whereas clock synchronization is a measure of the absolute agreement of said clocks. In the context of IEEE 1588 PTP, clock syntonization is not necessarily done using physical syntonization, but through clock adjustment strategies [1, p. 214]. As PTP does not enable continuous synchronization of clocks and instead provides periodic momentary synchronization events as illustrated in Figure 1, for one clock to be a useful representation of another clock, at least for some time after a synchronization event, the clocks must also be syntonized. Therefore, in the context of this paper, PTP clock synchronization also always implies PTP clock syntonization. Clocks which are synchronized and syntonized are said to be phase-locked, meaning that they increment time at the same rate and at the same time [2].

PTP defines a topology and ordering of distributed clocks (PTP instances) as part of a PTP *domain*. All clocks of a PTP domain are synchronized to the single *Grandmaster clock* of the domain [1, pp. 39–40]. Clock synchronization works through devices containing PTP clocks exchanging (communicating) PTP *messages* over PTP *communication paths* between a set of PTP *ports*, and capturing the timestamps of the reception or transmission of said messages [1, pp. 40, 74–76].

Clocks are generally divided into *ordinary clocks*, having a single PTP port, and *boundary clocks*, having multiple PTP ports [1, p. 41]. Ordinary PTP clocks can either be *master clocks*, being the source of time for all other PTP clocks on this PTP communication path, or *slave clocks*, synchronizing to the *master clock* on the respective PTP communication path. Boundary PTP clocks have multiple PTP communication paths associated and can therefore simultaneously be a PTP slave and master on different communication paths [1, pp. 47–48]. Such a PTP clock hierarchy is illustrated in Figure 2.

PTP works by communicating the current master time to a slave, and correcting for the master–slave message propagation delay at the slave. Therefore, two distinct types of messages are exchanged: *sync messages* and *delay measurement messages*. Sync messages communicate the current master timestamp to the slave. Since these messages experience the master–slave message propagation delay of the PTP communication path, the slave must further add this delay to the received master timestamp in order to recover the current master time. Therefore, the slave will issue delay request messages, which are answered by the master through delay response messages. An exchange of a sync-message and a pair of delay-request-message and response-message yields four timestamps, as described in Figure 1. From these timestamps the master–slave delay can be recovered as

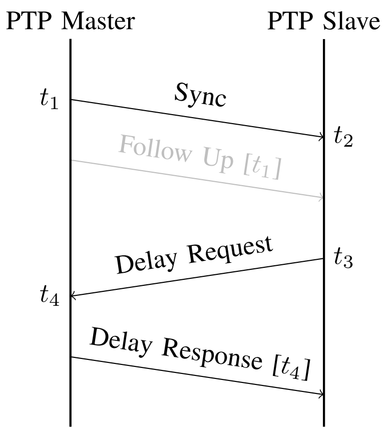

Fig. 1. An exchange of PTP Sync and Delay Request/Response Messages with associated timestamps.

$$d_{\text{master–slave}} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2},$$

and in turn the current master timestamp can be recovered as

$$t = t_1 + (t_{\text{slave}} - t_2) + d_{\text{master–slave}},$$

where t_{slave} is the current slave time to adjust for time passed since the sync message exchange [1, pp. 57–58].

Figure 1, in addition to the sync and delay measurement messages, further contains a *follow up* message. While a *one-step* PTP clock can embed the timestamp t_1 directly into the sync message itself, a *two-step* PTP clock can only accurately determine t_1 after the sync message has been sent. In this case, the follow up message is used to communicate the timestamp t_1 to the slave [1, p. 76].

PTP measures the master-slave message propagation delay using the accumulated delay of both, the slave-master and master-slave message paths (round trip delay), and subsequently assumes half of this delay is accurate to describe the master-slave message path. Therefore, PTP makes the basic assumption that the paths are entirely symmetric in terms of their delay, or the asymmetry is known [1, p. 371]. In practice this assumption may not hold, especially on non-direct physical links between PTP clocks. Intervening network elements feature buffers and other mechanisms which can impact path symmetry and delay variability. IEEE 1588 recommends using *transparent* clocks for intervening network elements, which correct for these additional delays in each propagated message. Furthermore, PTP may be used in a *peer-to-peer* delay measurement mode, as opposed to the *end-to-end* approach presented in Figure 1. In the *peer-to-peer* mode, the individual PTP link delays between two PTP ports are measured, as opposed to the total master-slave delay [1, p. 59]. This can further improve the accuracy of the delay estimation of the PTP communication path as a whole and reduce the influence of delay asymmetry of a link.

III. Problem Statement

The IEEE 1588 PTP time synchronization standard, as introduced in Section II, makes a range of assumptions about the software and hardware implementations of participating nodes in a PTP domain, in addition to the network structure [1, p. 39]. While some of these assumptions are critical to ensure reliable operation of the PTP protocol, such as the requirement of low PTP message loss, other assumptions are made purely to guarantee a high precision time synchronization: for instance, an offset in the acquisition of PTP message timestamps may result in an absolute time offset between slave and master clocks [1, p. 77].

In general, due to hardware and operating constraints, implementing PTP as described by IEEE 1588 along with all recommendations may not be feasible or practical. In particular, devices without special hardware capabilities for IEEE 1588 can be more attractive for the overall system compared to devices having hardware timestamping capabilities, or are already deployed in the field. Therefore, this thesis will analyze the effects of violating different assumptions made by PTP to match hardware and software constraints given by commodity microcontrollers.

To estimate the magnitude of error in time synchronization introduced by any such violated assumption, a controlled, open and measurable system shall be used to implement a subset of the PTP protocol. This system operates the PTP protocol and algorithms together with other PTP nodes using well-established and precise PTP-capable hardware and software. A measurement of the absolute synchronization error in different system conditions and configurations is subsequently used to deduce the effect of these system properties on the accuracy of PTP implementations in general. Furthermore, the ability to take system-internal measurements allows to gain confidence in the cause-effect relationship of different system properties and the observed synchronization accuracy. The system implementing the PTP subset and simulating different system conditions is further referred to as the Device Under Test (DUT).

IV. Analysis

The following section describes the identification and evaluation of important system properties, in order to design a suitable test and measurement system to acquire realistic and precise measurement data in accordance with the problem statement. In particular, decisions made in the design of this system are explained to provide a justification of the Device Under Test (DUT)'s assumed application area.

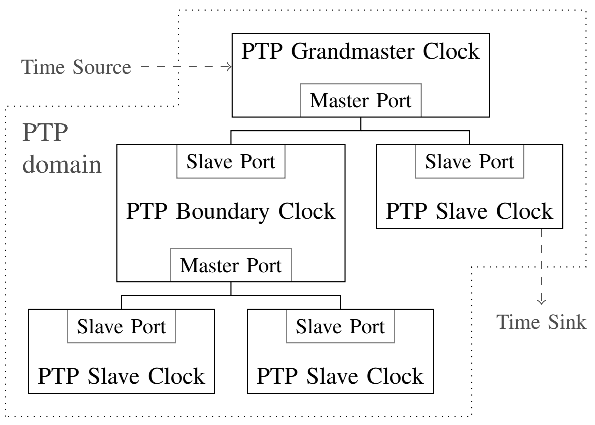

Fig. 2. An example of a PTP domain, outlining the clock hierarchy along different PTP nodes, respective PTP ports and communication paths.

A. *Identification of a Suitable PTP Clock Type*

In order to develop a realistic test scenario, a PTP mode of operation mirroring common real-world applications must be identified. Specifically, the type of PTP clock of the clock hierarchy (compare Figure 2) implemented by the DUT, along with other operating parameters, must be chosen such that typical use-cases for PTP are best described and generalized through the implementation of the DUT.

While generic microcontroller-based embedded systems may well take the role of a Grandmaster clock in a PTP domain, this is seldom expected to be the case. Most microcontrollers do not feature particularly stable local oscillators to serve as the basis for holding an accurate measure of time. Even with periodic re-synchronization to an external time source, such a system might experience significant clock drift. While this is the case regardless of the type of PTP clock implemented in such a system, the Grandmaster clock is expected to be a reliable and accurate source of time, and serves as an upper bound on the precision achievable by other clocks synchronizing to this source, with respect to an external time source. Not featuring hardware packet timestamping capabilities will further decrease the achievable synchronization precision [1, p. 489].

Compared to Grandmaster clocks, PTP boundary clocks are a more viable target for microcontroller-based embedded systems. Specifically for providing a time source to other transport protocols, such as industry-standard bus systems, microcontrollers can participate in a PTP domain via IEEE 802.3 Ethernet, and further distribute the timing information either through Ethernet, or other protocols using a Sink Adapter as specified in IEEE 1588 [1, p. 102, section 7.6.7.2].

Nonetheless, PTP slave clocks appear to be the most reasonable application area for generic microcontrollers within PTP. While only a single Grandmaster clock is required in a PTP domain, multiple slave clocks may synchronize to this Grandmaster, emphasizing the economical impact of using low-cost, low-power generic microcontrollers in these applications. Further, any errors introduced in the time synchronization of a PTP slave clock will not have a direct effect on the synchronization capabilities of other devices in the PTP domain. Finally, a PTP boundary clock encompasses a PTP slave clock as a sink, and a PTP master clock as a source of timing information. Therefore, results obtained as part of a PTP slave clock implementation should have a direct relation to the precision achievable with boundary clocks.

B. PTP Operating Parameters

In addition to the type of PTP clock implemented in the DUT, operating parameters of the PTP protocol also play an important role in the design of the test system and directly relate to results observed.

A PTP slave clock receives PTP sync messages from an elected master clock with a period T_s. Further, the slave clock must periodically initiate delay measurements by sending a delay-request message every T_d. Choosing these parameters can have direct influence on the observed synchronization accuracy. A higher synchronization period T_s causes the slave clock to have fewer accurate master-clock timestamps, and thus be decoupled from the master clock (*free-running*) for longer periods of time. A higher T_d can cause the slave to miss short-lived variations of the PTP communication path propagation delay. However, choosing either T_s or T_d too low will cause additional network and system load on participating PTP instances and PTP communication paths [1, pp. 103–104]. Both of these values are specified as powers of two (`logSyncInterval` [1, p. 103]; `logMinDelayReqInterval` for a minimum interval between two subsequent delay-request messages respectively [1, p. 104]). While default values vary across implementations[1], choosing reasonably high intervals can be useful to emphasize negative effects of such clock drift and the efficiency of compensation mechanisms.

Other parameters, such as the delay mechanism, the transport protocol (Ethernet, UDP/IPv4 or UDP/IPv6) and multicast or unicast operation are dependent on the test and measurement architecture, as elaborated in the following section.

C. Test and Measurement Architecture

To eliminate error sources outside of the DUT, it is important to choose an appropriate test and measurement architecture. Error sources outside of the DUT are an important concern for production systems, for example when operating PTP communication paths using (partially) non-PTP-capable active network equipment. However, the presence of external errors makes isolating system-internal error sources significantly more difficult. For this reason, the test architecture must be conceptually simple, with a minimal number of active and passive components on the time-critical PTP communication path between the master clock and slave clock.

To further provide a stable source of time for the slave clock to synchronize to, the master clock should operate in a *free-running* manner (that is, not synchronized or syntonized to any external clock source). As such, the provided timing information is obtained solely from the master clock's internal oscillator. By avoiding synchronization with external clock sources – such as Global Positioning System (GPS) – the master clock's frequency should experience a relatively constant frequency drift caused by physical properties of and environmental influences on the internal crystal oscillator alone, as opposed to frequent changes in both the frequency and frequency drift caused by adjusting to the external clock source. A basic assumption of this system is that the master clock uses a comparatively stable crystal oscillator with high short-term stability, ideally orders of magnitude more precise compared to the oscillator used by the DUT.

To precisely measure synchronization accuracy achieved by the DUT under different conditions, a multitude of options exist. For instance, on each complete PTP synchronization, the DUT may compare the current time estimate before the synchronization to the time corrected using the information of the synchronization message exchange. A major advantage of this approach is that no additional hardware or communication channel is required, which makes it ideal for estimating the accuracy of PTP systems in production. However, this in-system measurement can be subject to errors in the offset calculation. Furthermore, the synchronization performance might be affected by the presence of the in-system measurement itself. Lastly, an in-system measurement cannot verify correct compensation of the PTP communication path delay, as the device has no way to determine this parameter without cooperation of the elected PTP master clock through one of the delay measurement methods specified.

Instead, a synchronization signal, issued by both, the master and slave clocks at a fixed phase offset within a specific time interval, can be compared by an external system. A common example of such a signal is a per-second pulse, issued at the start of every second. This is commonly referred to as a PPS or 1-PPS (Pulse per Second) output, and can be found on many precision clock instruments. An external system can measure the difference in arrival time of these two pulses (phase offset), which directly correlates to the difference in time as kept by the two independent systems at that instant. If this pulse is generated in hardware without any software involvement, it will be unaffected by any other system-internal noise.

However, measuring clock synchronization precision using a Pulse per Second (PPS) signal issued by master and slave clocks also has significant disadvantages. For instance, measuring the difference between two PPS pulses only reveals the cumulative error of a clock, with no detailed information about the clock behavior within this period. The measured error can contain components of clock synchronization error, clock syntonization error, oscillator stability and oscillator noise. Hence, the measured error will be a composition of both internal errors and environmental influences such as temperature variations and crystal oscillator aging [5]. These error components can be both additive and subtractive. For instance, Figure 3 outlines a potential measurement deficit when using 1-PPS-style signals from two clocks: two fictional

[1]For instance, `ptp4l` of the Linux PTP Project uses $T_s = 2^0 = 1\,\text{s}$ by default [3], whereas Juniper devices use $T_s = 2^{-6} \approx 16\,\text{ms}$ [4, p. 88].

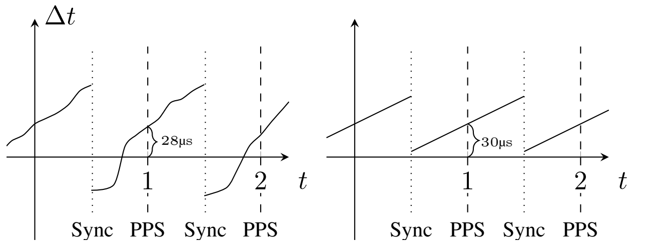

Fig. 3. Cumulative synchronization measurement error with 1PPS-style measurements.

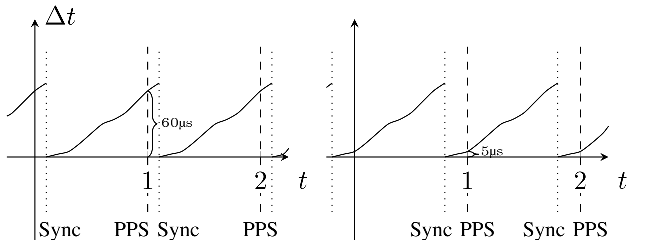

Fig. 4. Biased synchronization measurement through synchronization phase offset.

clocks are depicted, synchronized through a sync message, and issuing a PPS signal each second. Both clocks show inaccuracies, whereas the left clock is significantly less stable than the right. The graph depicts the difference in time as held by these clocks. For an outside observer of only the two PPS signals, the left and right cases cannot be reliably distinguished.

Furthermore, only acquiring one data point per second requires long-running measurements to gain statistical confidence with respect to observed effects.

Finally, synchronizing two clocks and measuring the accuracy through a PPS signal introduces the risk of measurement bias: depending on the phase offset of the synchronization messages within a given second, the clock will be free-running and subject to its noise and drift characteristics for longer or shorter periods of time, respectively. For example, when synchronizing the absolute time at the end of a given second (immediately prior to issuing a PPS pulse), the clock's syntonization and noise will have an insignificant effect on the observed result, compared to when synchronizing at the start of a given second. This is illustrated in Figure 4, showing two identically behaving clocks with different sync message offsets and corresponding measurement bias.

D. DUT Hardware

To be able to acquire measurement data as described in the problem statement and elaborated in the previous sections, an appropriate hardware platform for the DUT must be used. In particular, the ability to take measurements (such as the PPS output) without software involvement is required. Further, to reproduce realistic conditions as found in commodity microcontrollers, the DUT should be architecturally similar to such systems. Finally, using PTP imposes a set of strict requirements on the system, such as a compatible transport or a locally running clock [1, pp. 45, 74].

Even though this paper primarily analyzes the effect of using PTP without proper hardware timestamping support, the DUT should feature PTP hardware timestamping capabilities in order to establish a measurement baseline. This baseline measurement will be used to test the software PTP implementation correctness, and is used as a best case scenario to compare other measurements to.

In general, these basic requirements are fulfilled by many modern commodity microcontrollers (based on processor architectures such as ARM Cortex-M), featuring IEEE 802.3 Ethernet and IEEE 1588 timestamping capabilities. However, microcontrollers in general cannot be introspected internally, as to determine where delays and jitter in the processing of internal events and signals occur. Furthermore, manufacturers often supply only a high-level description and overview of the microcontroller's internal structure.

In contrast to that, a Field Programmable Gate Array (FPGA) can be used to synthesize a complete SoC, with an architecture similar to current microcontrollers. However, by being entirely re-programmable to synthesize any logic circuit within the physical limitations of the FPGA, such a platform features significantly more flexibility and introspectability compared to fixed-circuit microcontrollers. For example, internal signals of the synthesized SoC can be routed to external pins of the FPGA to take precise timing measurements. Furthermore, custom hardware (*cores*) can be created to offload measurement and computation tasks which would otherwise need to be done in software on commodity microcontrollers. Depending on how the required IEEE 802.3 transport is integrated with the FPGA, hardware timestamping functionality can be implemented within the synthesized logic in the FPGA itself.

E. PTP Clock Implementation

Conventional microcontroller platforms typically do not feature a precision real time clock, but instead provide at least a single system timer module. Examples for this are the system timer (*SysTick*) found on ARM Cortex-M, or the machine timer (*mtime*) of RISC-V systems adhering to the RISC-V Instruction Set Privileged Architecture [6, p. 32].

These clock implementations either do not support adjusting the current clock time as a hardware constraint, or the clock's current value cannot be adjusted without violating invariants of other parts of the system, given that software components rely on the fact that the timer value will be monotonically increasing. Since this single counter must be shared by many software consumers running on the same hardware through timer virtualization techniques, this is almost always the case.

This is in contrast to typical hardware-assisted PTP implementations. Here, an *adjustable timer* is provided, implementing a free-running counter which can be set to an arbitrary value and frequency-adjusted to run synchronized to a master clock to provide an accurate local time reference. Notably, this local clock is further used to timestamp received and transmitted PTP frames.

Microcontrollers may feature an adjustable timer module which supports setting an absolute value, as well as adjusting the timer frequency to synchronize to a distinct master clock. However, for the reasons outlined above and to avoid enforcing specific hardware requirements, this clock must be assumed to be independent of any timestamp reference clock, either in the MAC controller (hardware-assisted) or a standard system clock (software-based).

Therefore, two scenarios must be considered:

1) The microcontroller does feature an adjustable clock module which supports setting the current time, as well as adjusting the operating frequency up to a certain degree. It does not need to run monotonically increasing. This hardware clock can be used as a local reference clock, synchronized to the PTP master clock. It cannot be used for timestamping PTP packets.
2) The microcontroller does not feature an adjustable clock module. A local, non-adjustable system clock must be used as a basis to provide a local reference clock synchronized to the PTP master clock through clock virtualization techniques. When timestamping transmitted and received PTP frames in software, the gathered timestamps can be assumed to be relative to the system clock frequency.

While the second approach (virtualizing a synchronized real-time clock over a monotonic free-running system clock) can be implemented in virtually any system, this implies translating the system clock into the real-time clock using software operations. Depending on the frequency-adjustment mechanism and implementation strategy, this is not always feasible with respect to the resource constraints, especially on microcontrollers without hardware multiplication and division support. Consequently, if an appropriate adjustable clock is present, it should be used as a local reference clock.

In the context of this work, using an appropriate adjustable clock hardware module is required. This is because the 1-PPS signal, as required by the Test and Measurement Architecture Analysis of Section IV-C, must be generated independent of the DUT's software stack to avoid system-internal influences on the observed result. This can only be achieved by a second, free-running counter which can be set, modified and adjusted in frequency. When the current timer value $\bmod 1\,\mathrm{s}$ reaches 0, a 1-PPS pulse must be generated accordingly, without any software involvement.

F. DUT Software

The hardware platform as described in the previous section must be paired with software compatible to the hardware platform and capable of operating the PTP protocol. Furthermore, the DUT must be able to simulate desired measurement workloads. There are virtually endless options for implementing such a software stack. Conceptually, embedded systems are hard to generalize: systems can span from having resources just sufficient to implement simple control loops, to running full general purpose Operating Systems (OSs) and workloads [7, p. 95]. Hence, it is difficult to define one single software platform and architecture as representative for the broad range of embedded devices and their use cases.

Nonetheless, specific characteristics of the implemented software stack can have effects on both, the implementation complexity of PTP, as well as the software timestamp accuracies achievable. Therefore, it is important to identify these relevant characteristics, to be able to estimate their effect on the measured results.

A basic classification of embedded devices is presented in [8]: embedded devices are classified into *general purpose OS-based devices*, *embedded OS-based devices*, and *devices without an OS-abstraction*. This classification can be used to perform a preselection of the DUT software stack architecture.

Devices without an OS-abstraction typically use a single control loop and receive interrupts triggered by peripherals. These software systems can execute full control over the underlying device, which can be used to meet different design goals, for instance to react to incoming events quickly. However, such a highly flexible system architecture also implies that is does not necessarily have clearly distinguishable characteristics or follow established OS properties. A requirement of this paper is to be generalizable to other embedded systems, and as such this class of embedded system is a poor fit as a representative DUT.

On the other end of the spectrum, there are general purpose OS-based embedded systems. These generally run commodity OSs and hardware, retrofitted to be suitable for the required embedded use case. The high complexity, generic nature and high hardware requirements of such systems make introspecting the OS behavior difficult. Furthermore, the requirements posed by many general purpose OSs are in contrast to the motivation to use low-cost generic microcontrollers for implementing PTP.

Embedded OS-based systems strike a balance between the lower resource requirements of devices without an OS-abstraction and those using general purpose operating systems. They define abstraction layers, can schedule multiple distinct tasks and use established operating system concepts, while being able to run in low-resource environments. The ability to use internal abstraction layers for implementing PTP on a set of defined Application Programming Interfaces (APIs) to lower layers, as well as the possibility to run synthetic workloads either in parallel or sequentially to the PTP implementation makes these systems viable for implementing as part of the DUT.

Common properties of embedded OSs that can significantly influence the real-time characteristics of the system are the scheduling algorithm and its parameters, interrupt latency in software and context switch overhead [9]. Thus also the accuracy and stability of acquired software timestamps is influenced as a result of these characteristics.

V. Architecture

Following the analysis of Section IV, which identifies important properties and characteristics of a measurement

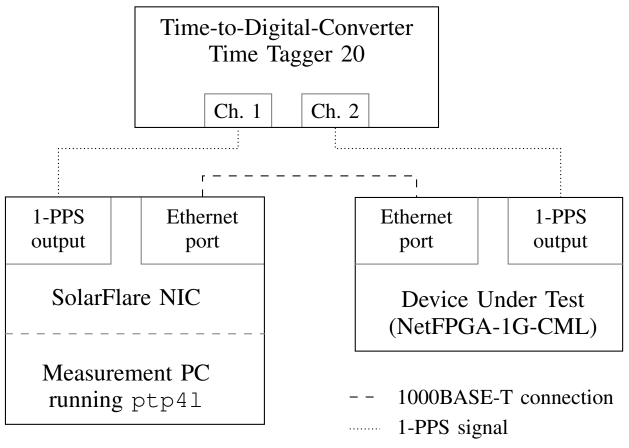

Fig. 5. Structured overview of the system model, including the DUT FPGA, the PTP Grandmaster Ethernet NIC and the TDC measurement device.

architecture, this section will elaborate on the details of the final DUT and measurement setup. An overview of the system and measurement architecture is depicted in Figure 5.

A. PTP Architecture

The system architecture encompasses a single PTP domain. As discussed, the Device Under Test implements a PTP slave clock. Furthermore, there exists a Grandmaster clock for the PTP slave to synchronize to. No other devices are placed on the PTP communication path in between the Grandmaster and slave clock devices. The devices are interconnected through a short ($< 5\,\mathrm{m}$) 1000Base-T Ethernet link. This is to avoid any external influences and unpredictable delays on the transmission of PTP messages. The PTP communication itself, in accordance with these constraints, is using a multicast-based IPv4 User Datagram Protocol (UDP) transport. Other transport mechanisms, such as unicast transmission, IEEE 802.1 packets or IPv6 UDP packets are also compatible with this system architecture and should not influence the observed results.

The Grandmaster PTP device is implemented using a commodity x86-64 Personal Computer (PC) running a Linux distribution with kernel version `5.4.108`. It features a Solarflare Communications `SR203 SF10-050020` Ethernet network adapter, which has built-in support for IEEE 1588 hardware timestamping. Importantly, this network adapter features a PPS output signal, issuing a pulse on precisely the start of a second with respect to the network adapter's internal clock. The Linux PTP Project's `linuxptp`[2] PTP implementation is used to implement the Grandmaster's PTP software stack, which in turn uses the hardware timestamping capabilities of the Solarflare network adapter.

Notably, the `linuxptp` software is configured to have a `logSyncInterval` of 0, resulting in one PTP sync message per second. However, this causes issues of measurement bias as described in Section IV-C and illustrated in Figure 4: by always synchronizing at the same phase offset within one second, depending on the start of the `linuxptp` software, the DUT will be free-running for shorter or longer periods of time respectively. While the phase-offset of the sync messages experiences long-term drift of approximately $1\,\mathrm{s}$ within $8\,\mathrm{h}$, this still introduces systematic errors. To reduce the impact of such measurement bias, the `linuxptp` software is patched to always issue sync messages with a delay of $1010\,\mathrm{ms}$, causing the phase offset to wrap around every $\frac{5}{3}\,\mathrm{min}$. This will distribute the effects of a shorter and longer free-running local clock of the DUT over the measurement interval.

B. DUT Hardware

The DUT, implementing the PTP slave clock and measurement workloads, is built on an FPGA platform. The precise FPGA used is expected to only have limited influence on the observed results, as long as the specified logic circuit (*Gateware*) can be synthesized for and thus represented on the FPGA, and all timing constraints are met. For completeness, the specifications are provided nonetheless: for this project, a NetFPGA-1G-CML board containing a Xilinx Kintex-7 XC7K325T FPGA is used. It features RTL8211 IEEE 802.3 Ethernet transceivers (*PHYs*), which do not have any IEEE 1588 timestamping capabilities by themselves. The FPGA is connected to an external $200\,\mathrm{MHz}$ low-jitter oscillator [10].

The logic synthesized for the FPGA must include all aspects required for a software stack as described in Section IV-F. This includes a Central Processing Unit (CPU), on which the embedded OS along with the PTP software stack and synthetic workloads can be executed. Furthermore, it must be able to drive required peripherals, such as the attached RTL8211 IEEE 802.3 Ethernet transceivers. Finally, it must include some local clock as mandated by IEEE 1588 and feature at least a PPS output to be able to measure the synchronization accuracy against the PTP Grandmaster [1, p. 38].

To generate the FPGA logic description in a Hardware Description Language (HDL) that can be used to synthesize a so-called *bitstream* for the FPGA, the free-software project *LiteX* is used. LiteX is a SoC builder and can be used to generate full FPGA designs, incorporating *cores* such as CPUs, timers or other peripherals [11]. Different parts of the design are interconnected using a bus protocol, such as *Wishbone*[3]. Logic designs generated by LiteX, incorporating a CPU, memory and other peripherals, are architecturally sufficiently close to commodity microcontrollers.

The LiteX design describing the DUT hardware is centered around a *VexRiscv*[4] CPU core, implementing the RISC-V $32\,\mathrm{bit}$ integer instruction set with support for hardware multiplication and compressed ($16\,\mathrm{bit}$) instructions, referred to as `rv32imc`. All components of the system are interconnected through a $32\,\mathrm{bit}$-wide Wishbone bus. Other subsystems connected to this bus include a $1\,\mathrm{MB}$ SRAM memory synthesized on the FPGA, a Universal Asynchronous Receiver Transmitter (UART) core for serial communication, a $64\,\mathrm{bit}$-wide hardware timer and a *LiteEth* Gigabit Ethernet core.

[2]http://linuxptp.sourceforge.net/

[3]https://cdn.opencores.org/downloads/wbspec_b4.pdf

[4]https://github.com/SpinalHDL/VexRiscv

The system, including all timers, is driven by a $100\,\mathrm{MHz}$ clock signal generated through a Xilinx Mixed-Mode Clock Manager module based on the external $200\,\mathrm{MHz}$ precision oscillator on the NetFPGA-1G-CML board.

Furthermore, the DUT hardware features an *adjustable clock* peripheral, as described in Section IV-C. This peripheral is implemented using a counter driven by the system-global $100\,\mathrm{MHz}$ clock signal. It exposes a register-based interface to the VexRiscv-CPU, which can be used to either set (using absolute values) or modify (using relative values) the current timer value. Furthermore, the CPU can instruct the module to adjust the frequency of the clock through a $32\,\mathrm{bit}$ signed fractional frequency control value. For example, setting the frequency adjustment to 3 would cause the clock to operate at $1 + \frac{1}{3} = 1.\overline{3}$ times its reference oscillator, whereas setting -4 would cause the clock to operate at $1 - \frac{1}{4} = 0.75$ times its reference oscillator. The CPU can further request an electrical pulse to be issued when the current timer value is evenly divisible by some divisor. The timer will not skip or issue duplicate pulses because of a fractional frequency adjustment.

C. DUT Software

The synthesized SoC must further be matched with software, in accordance with the analysis of Section IV-F. In particular, it must be compatible with the synthesized hardware and implement a PTP software stack. For this research project, the Tock embedded operating system[5] has been ported to work on LiteX SoCs with a VexRiscv CPU. Tock is a novel embedded operating system targeting low-power and low-resource microcontrollers. It can run on systems having $48\,\mathrm{MHz}$ clock speed and featuring as little as $64\,\mathrm{kB}$ of main memory. Tock consists of a monolithic kernel and unprivileged userspace applications, interacting through system calls. One of its key distinguishing features is its use of language-based and type-system-based isolation mechanisms in the kernel. This is achieved by writing the kernel almost exclusively in the Rust programming language, and utilizing Rust's fundamental safety and ownership models, and type system. This means that significant parts of the kernel code must be trusted only to eventually give control back to the core kernel, and are otherwise limited by the bounds provided through the type system. This reduces the risk of vulnerabilities caused by implementation errors. Examples include working with variable-length buffers provided by applications or networked systems, which frequently lead to buffer overflows in other languages [12]. The unprivileged userspace applications are isolated through hardware isolation mechanisms, such as the RISC-V Physical Memory Protection (PMP) module. The Tock kernel is cooperatively scheduled and has full control over the hardware, whereas the userspace is preemptively scheduled by the kernel. As Rust is used to implement the Tock kernel, no Garbage Collector (GC) is used, enabling deterministic system behavior [13], [14].

Because of these characteristics, Tock represents a reasonable choice for the DUT software stack. Especially the low resource requirements, together with the ability to run multiple mutually distrustful applications within a single such system emphasizes the economic advantage of using such an OS to implement a PTP slave clock device. In addition to that, the deterministic, cooperatively scheduled and privileged kernel written in Rust provides a viable target for implementing time-critical parts of the PTP software stack. Further, Tock meets the requirements of Section IV-F, is a full embedded OS and supports running multiple workloads in a pseudo-parallel fashion [13].

Tock does not currently have a full TCP/IP stack present in the kernel, and does not implement protocols required to establish socket-based communication with other devices in a Local Area Network (LAN) network, such as Address Resolution Protocol (ARP) or IPv6 Neighbor Discovery (ND), Internet Protocol (IP) and UDP. Instead, to allow communication of PTP messages and other common LAN protocols, a *TAP*-like driver for implementing a Layer-2 IEEE 802.1 transport in userspace is implemented as part of this work. This TAP-interface is in turn connected to the hardware IEEE 802.3 Ethernet MAC driver for the DUT's *LiteEth* Ethernet MAC. Such a setup enables userspace applications to implement a full TCP/IP network stack and communicate with other network devices using commodity protocols like ARP, Internet Control Message Protocol (ICMP) and UDP. For implementing the TCP/IP protocol stack in userspace, the free and open source LwIP library[6] is used. While a TCP/IP stack implemented in the Tock kernel might be significantly more efficient compared to an implementation as part of a userspace application, this strategy requires far less implementation effort. By acquiring and associating timestamps to incoming and outgoing IEEE 802.1 frames in the kernel and passing them to the LwIP network stack, implementing the packet processing in userspace should not have any impact on the precision of the acquired timestamps.

Whereas a userspace application is used to manage various aspects of the DUT network communication, many core aspects of PTP are implemented within the Tock kernel as untrusted (*capsule*) code. This is to minimize the interactions across the userspace–kernel boundary during the operation of the PTP state machines and to use the predictable timing characteristics of the Tock kernel for time-critical sections of the PTP implementation. For instance, because of the aforementioned hardware memory isolation mechanisms, setting a hardware clock memory-mapped register from userspace would require a system call into the Tock kernel, which can introduce unpredictable delays [13].

Notably, as an attempt to compensate potential errors in the acquisition of PTP message timestamps, the PTP implementation as part of the Tock kernel further includes a PTP message *low-pass filter*. This filter uses a metric defined on incoming PTP messages along with their timestamps and, depending on

[5]https://tockos.org/

[6]https://savannah.nongnu.org/projects/lwip/

whether the measurement is deemed in bounds with respect to this metric, forwards the PTP messages to the PTP stack for further processing. For sync-messages, the metric is defined on a pair of subsequent messages, as the ratio of the increase of the reported master time and slave time of the slave device's monotonic local clock:

$$m_{\text{sync}}(\mathrm{a},\mathrm{b}) = \frac{\text{b.master\_time} - \text{a.master\_time}}{\text{b.slave\_time} - \text{a.slave\_time}}.$$

For delay-request-messages and delay-response-messages, the metric is defined on a pair of complete delay measurements, through the ratio of the increase of the master receive times and slave send times:

$$m_{\text{delay}}(\mathrm{a},\mathrm{b}) = \frac{\text{b.master\_receive} - \text{a.master\_receive}}{\text{b.slave\_send} - \text{a.slave\_send}}.$$

These values are compared to the last n sync-messages or delay-measurements respectively, with the highest and lowest m deviations from the mean filtered out. Thus, if the current measurement has a comparatively high deviation from the mean observed metric over the past m measurements, it is not considered for further processing.

D. Synchronization / Syntonization Accuracy Measurement System

As discussed in Section IV-C, the DUT's synchronization and syntonization accuracy shall be evaluated by comparing the PPS output of the PTP Grandmaster and PTP slave (DUT) devices. Both devices issue a 1-PPS pulse on the start of each second. The delay between these pulses, as well as the variance of these delays, can provide information about the synchronization and syntonization of the DUT's PTP slave clock to the Grandmaster's PTP clock.

To automatically measure these delays over a long-running measurement series, a Swabian Instruments Time Tagger 20 device is used. This device is a so-called Time-to-Digital-Converter (TDC), being able to measure the delay between incoming events (pulses) with a RMS jitter of $34\,\text{ps}$. The resulting *Time Tag Stream* can be sent to a PC for further analysis, for example autocorrelation of events occurring on two input channels [15].

Therefore, the measurement system as seen in Figure 6 consists of the Time Tagger 20, the Grandmaster PTP clock and the DUT. The Grandmaster is connected over a standard, short 1000BASE-T link as described in the previous sections. Furthermore, both the Grandmaster PTP clock's 1-PPS output, as well as a digital output of the DUT FPGA are connected to an input channel on the Time Tagger 20 respectively. It should be noted that an amplifier circuit is used in between the Grandmaster PPS output and the Time Tagger 20 input to support the resistive load ($50\,\Omega$ line termination) imposed by the Time Tagger device. This adds a constant delay to pulses issued by the Grandmaster PTP device, which is below $1\,\mu\text{s}$ as verified using an oscilloscope, but must be accounted for.

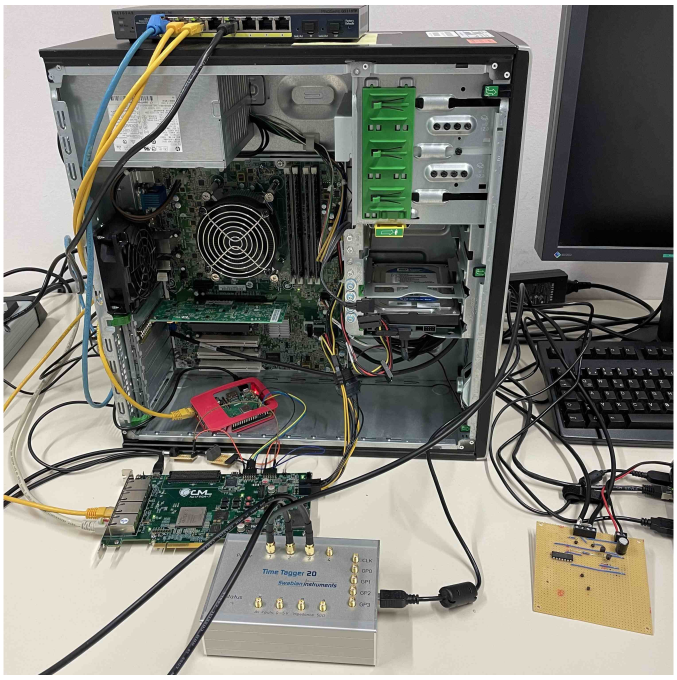

Fig. 6. Photo of the complete measurement system. In the front, the Time Tagger 20 Time-to-Digital-Converter is connected to both the NetFPGA-1G-CML FPGA card behind it, as well as to the Solarflare Communications NIC mounted in a PCIe slot of the measurement computer. The PPS output of the Solarflare NIC is amplified by the circuit on the right hand side to support the line-termination resistive load imposed by the Time Tagger 20. The FPGA outputs connected to the Time Tagger have a $330\,\Omega$ resistor placed in series to reduce the current provided by the FPGA.

VI. Statistical Evaluation of Clock Stability and Synchronization / Syntonization Accuracy

To properly evaluate the synchronization and syntonization accuracy of the two PTP clocks as described in Section V-D, proper statistical methods must be selected. Furthermore, Section IV-C describes that the synchronization and syntonization measurement architecture works under the assumption that, compared to the DUT's PTP slave clock oscillator, the PTP Grandmaster uses a comparatively stable oscillator, with high short-term stability. This assumption shall be verified to ensure the statistical relevance of measured PTP slave clock synchronization and syntonization accuracy.

A well defined, periodic delay measurement of two synchronized and syntonized independent clocks, such as a time-correlation of 1-PPS outputs, yields only a cumulative error of various influences. This has been discussed in Section IV-C. With the goal of this paper being to analyze the feasibility of using commodity microcontrollers for time synchronization through IEEE 1588 PTP, it is important to isolate these different error sources. As their individual effect cannot be observed directly, other statistical methods have to be used to acquire estimates of the quantitative influences of these effects on observed results.

A naïve approach to estimate clock stability behavior of one clock with respect to another could be to use the standard variance or deviation of the delays between any two observed and correlated 1-PPS pulses. However, the standard variance is based on expressing the expected deviation from the average. In the case of two clocks this would be the average difference in time passed between two subsequent events of the two clocks. Though, this average is not necessarily stable over time, for example due to clock drift, and as such the standard deviation does not converge for some noise types [16]. Because long-term divergence of two clocks is in general reduced or prevented when using proper synchronization mechanisms, these characteristics primarily affect the measurements of long-term stability and accuracy of free running clocks.

Instead, the two-sample variance or Allan variance and the corresponding Allan deviation overcome these issues by using the differences of subsequent frequency deviations. The two-sample Allan variance is defined as

$$\sigma_y^2(\tau) = \frac{1}{2(M-1)} \sum_{i=1}^{M-1} [y_{i+1} - y_i]^2,$$

with y_i being the ith of M fractional frequency values averaged over the measurement interval τ [16]. The Allan variance can further be extended to the *overlapping Allan variance*, which uses all available overlapping samples at each averaging time τ to gain confidence in the observed two-sample Allan variance estimate.

However, the Allan variance and its derivative variances only give an estimate of a clock's frequency or time stability behavior with respect to a reference clock. Therefore, it may well be the case that the reference clock itself can be the source of at least a part of the observed short-term and / or long-term frequency / time deviation [17]. However, in [17] Gray and Allan present a method (coined *Three Cornered Hat* method) to estimate instabilities of each clock in a set of three clocks [18]: assuming three independent oscillators, $O = \{A, B, C\}$, the frequency stability of each pair of oscillators can be measured: $\{\sigma_{XY}^2 \mid \{X, Y\} \in \binom{O}{2}\}$. As further described in [17], by following the assumption that the oscillators are independent and thus removing correlation terms, the equation for the Allan variance (and its derivates) can be written as shown in [18]:

$$\sigma_{XY}^2(\tau) = \sigma_X^2(\tau) + \sigma_Y^2(\tau).$$

Given a sufficient number of pairs of independent oscillators, these equations can be solved for a particular oscillator yielding, for example:

$$\sigma_X^2(\tau) = \frac{1}{2}(\sigma_{XY}^2(\tau) + \sigma_{ZX}^2(\tau) - \sigma_{YZ}^2(\tau)).$$

In accordance with this method, the three oscillators involved in the measurement setup described above – namely the Grandmaster PTP clock, slave / DUT PTP clock and the Time Tagger 20's internal time base – are measured against each other.

The results obtained by using the Three Cornered Hats method with the Overlapping Allan Deviation to estimate the clock stability characteristics of the clocks as part of the measurement setup are shown in Figure 7 in a log–log graph.

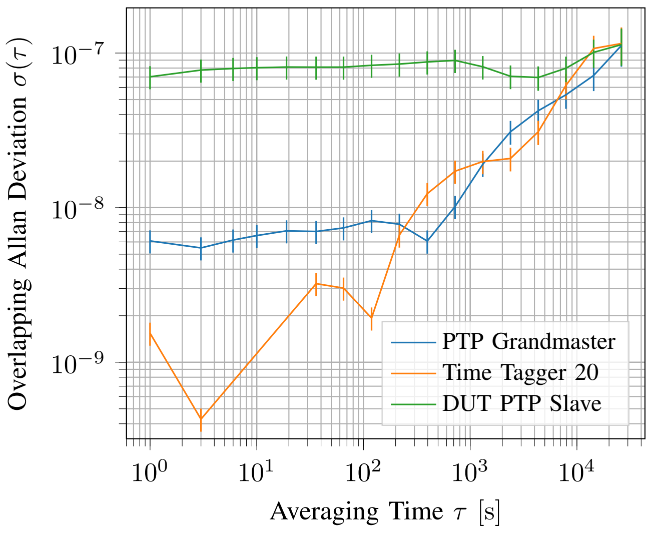

Fig. 7. Clock stability of the individual clocks in the measurement setup. Calculated by the Three Cornered Hats method using the Overlapping Allan Deviation. Error bars display 50σ.

With increasing τ, the calculated Allan Deviation represents stability behavior for longer intervals of time. The measurements displayed have been obtained by first synchronizing and syntonizing the DUT PTP slave clock to the PTP Grandmaster for 30 min, and then disconnecting the PTP communication path between the DUT and the PTP Grandmaster. The devices were subsequently running independently for 24 h, with the PTP Grandmaster and slave issuing a 1-PPS signal respectively. These pulses have been recorded with respect to the Time Tagger's internal time base, which allows to reconstruct the time differences between pulses of the PTP Grandmaster and slave devices.

The graph of Figure 7 shows that both the PTP Grandmaster and the Time Tagger 20 time base have a significantly lower – at least around one order of magnitude – short-term deviation characteristics compared to the DUT's internal oscillator. However, all clocks seem to be approximately equally stable when examining long-term behavior. Furthermore, the graph shows defects in the results: for some τ in the Time Tagger 20's estimation, in the graph visible for $\tau \in \{6, 10, 19, 36\}$, $\sigma(\tau) < 0$. Such imaginary Allan Deviations are not physically possible. According to Premoli and Tavella, violation of positiveness in these inferred variances can be attributed to (i) non-negligible uncertainty in the measured time differences due to the measurement device, (ii) non-temporary or pseudo-contemporary measurements of different pairs of clocks, (iii) insufficient measurement samples, or (iv) violation of the hypothesis that clocks are uncorrelated [19]. Given (i) the measurement device is built to precisely measure significantly smaller time differences, (ii) the measurement series of all pairs of devices are recorded with respect to a common time base, and (iv) each measurement consists of $24\,\text{h} * 60\,\text{min} * 60\,\text{s} = 86400$ data points, (iv) is the most likely explanation. The hypothesized correlation can be explained through two observations:

1) The three clock devices are electrically interconnected and reside in the same environment. Therefore, they are subject to common environmental influences, such as the surrounding temperature. Two devices (the Grandmaster Ethernet network adapter and the DUT FPGA board) are further powered by the same commodity PC power supply unit.
2) While the time deviation of the Time Tagger and one of the PTP clocks, issuing a 1-PPS pulse, can be measured by only involving two devices – the respective clock and the Time Tagger itself – this is not possible for measuring the deviation between the PTP Grandmaster and slave devices. The 1-PPS pulses issued by both devices are recorded by the Time Tagger with respect to its internal time base. This can be used to mathematically reconstruct the time difference between those two pulses. However, this reconstruction uses the Time Tagger's time base, and thus may leak the Time Tagger's internal oscillator behavior into this measurement series.

Even with these imprecisions and defects in the clock stability estimates, the observed results confirm the conjecture of the PTP Grandmaster clock having high short-term stability compared to the PTP slave clock, as postulated in Section IV-C. This ensures that the PTP slave clock synchronization and syntonization accuracy will not overproportionally suffer because of a poor short-term clock stability of the PTP Grandmaster.

For measuring time stability in a time distribution system, such as a PTP domain, Riley and Howe recommend using the *time Allan variance* $\sigma^2_{\text{tdev}}(\tau) = \frac{\tau^2}{3}\sigma^2_{\text{mdev}}(\tau)$ and its respective *time Allan deviation* $\sigma_{\text{tdev}}(\tau) = \sqrt{\sigma^2_{\text{tdev}}(\tau)}$. The time Allan variance is defined based on the *modified Allan variance* as defined in [16, p. 17]. The modified Allan variance is an extension of the Allan variance, including an additional phase averaging operation. The time Allan deviation has a distinct advantage of being equal to the standard deviation for white phase modulation noise [16]. Furthermore, while other Allan variances are dimensionless, the time Allan deviation is expressed in seconds, allowing for quantization of the expected synchronization error in a time distribution system [20]. Thus, σ_{tdev} is to be used as a measure for estimating the synchronization accuracy between the PTP Grandmaster and slave devices.

However, the time Allan deviation $\sigma_{\text{tdev}}(\tau)$ for the entire range of $\tau \in [1\,\text{s}; t]$, where t is the complete measurement duration, is not interesting. Because of the fact that the PTP clocks are synchronizing, $\lim_{\tau\to\infty}\sigma_{\text{tdev}}(\tau) = 0$, as proper synchronization effectively prevents long-term drift. This is illustrated in Figure 8 by an example plot of $\sigma_{\text{tdev}}(\tau)$ for $\tau \in [1\,\text{s}; 6\,\text{h}]$ of two synchronizing clocks. Rather, the short-term clock deviation ($\tau \approx 1\,\text{s}$) characterizes the clock synchronization and syntonization accuracy. As the (time) Allan deviation compares subsequent fractional frequency differences Δy_i averaged over the measurement interval τ, for synchronizing clocks and $\tau \approx 1\,\text{s}$, $\sigma_{\text{tdev}}(\tau) \overset{\propto}{\sim} \sigma_{\text{stddev}}\{\Delta p_j\}$,

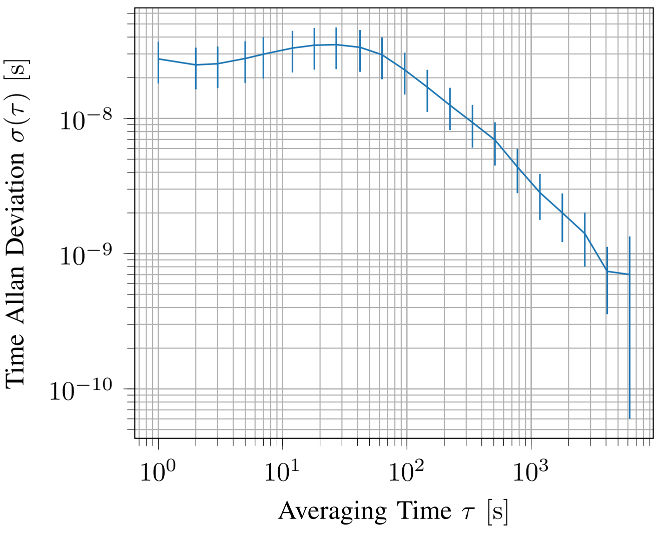

Fig. 8. Time Allan deviation over differences between corresponding 1-PPS pulses of the PTP Grandmaster and slave clocks (phase offsets), while they are synchronizing. The time Allan deviation converges to 0 because of the continued synchronization. Error bars display 50σ.

Δp_j being the time difference between corresponding jth 1-PPS pulses issued by both clocks. This is confirmed by Riley and Howe, stating that for white phase noise the time Allan variance is equal to the standard variance [16, p. 18]. Thus $\sigma_{\text{tdev}}(\tau \approx 1\,\text{s})$ expresses the sum of oscillator errors in both the PTP Grandmaster and slave clocks since the last synchronization up to the 1-PPS pulse generation, as well as errors introduced as part of the synchronization and syntonization. Hence, this measure will be used as the primary indicator for PTP synchronization accuracy. Further characteristic defects in the clock synchronization behavior may be analyzed by other statistical instruments and visual inspection of the data.

VII. Evaluation of Measurement Data

In accordance with the previous sections and the problem statement of Section III, the synchronization behavior and accuracy between the PTP Grandmaster and DUT PTP slave systems is to be measured.

A. *Measurements overview*

All measurements are conducted over a period of $6\,\text{h}$, which leads to approximately 21600 1-PPS pulses issued by both the Grandmaster and slave PTP clock devices. The offset between these pulses is recorded by the Time Tagger 20 device using an auto-correlation measurement series. These phase offsets are used to reconstruct the fractional frequency differences, as required for the calculation of the Allan variance.

In general, each measurement is based on a set of software configuration options of the DUT, hardware configuration options and simulated external influences. Internal variables for the device under test include:

1) the implemented PTP timestamp point. This is the point where a timestamp of a given PTP message, for ingress as well as egress, is acquired within the DUT.

2) whether or not a software low-pass filter is applied to acquired PTP message timestamps to detect inaccurate measurements.
3) simulated system loads through multiple applications running simultaneously.
4) whether syntonization is enabled, or PTP only performs clock synchronization.

To simulate external events to be handled by the device, a *random interrupt generator* core is added to the FPGA design. This core uses a Linear Feedback Shift Register (LFSR)-based Pseudorandom Number Generator (PRNG), producing the bit sequence `PRBS31` with polynomial $x^{31}+x^{28}+1$, being statistically random within the sequence length of 2 147 483 647 bit. The DUT can instruct the random interrupt generator to reduce the statistical frequency of interrupts relative to the SoC's clock speed, by use of a frequency division register. The frequency divisor is tuned such that approximately 1500 interrupts will be issued per second. The interrupts are not queued, so the DUT's OS must acknowledge each interrupt prior to the next being issued.

Furthermore, artificial system load is simulated by running an additional process on the Tock OS, next to the application implementing the LwIP TCP/IP stack processing PTP packets. This additional application continuously executes the algorithm by Dik T. Winter to calculate π to 800 decimal digits[7]. This application imposes computational load on the system and produces a large amount of system calls and serial output.

The implemented PTP timestamp point is suspected to have a large influence on the overall achievable synchronization accuracy and makes the device susceptible to other error sources within the system. Assuming that the hardware-based Ethernet MAC packet handling has constant time characteristics, it should still be reliable regardless of other system behavior. However, chip interrupt handling, the OS kernel architecture, and process scheduling can all introduce significant additive second-order deviations from the PTP reference message timestamp point [1, p. 79]. Other system design choices, as well as the system's environmental influences such as interrupt frequency, can also have a significant influence on the achievable synchronization precision. Hence, the combination of various implemented PTP timestamp points across the system architecture, along with some variation of other system properties is expected to provide sufficient information to derive conclusions about the achievable precision of PTP in generic microcontroller systems with different system properties.

B. Measurement results

The relevant measurement series' time Allan deviations for small τ can be seen in Figure 9. Furthermore, the measurement series are described in Table I, along with their primary system characteristics, as well as overlapping Allan deviation values for specific values of τ. In addition to that, Table I also lists the average time difference between two correlated 1-PPS pulses

[7]https://crypto.stanford.edu/pbc/notes/pi/code.html

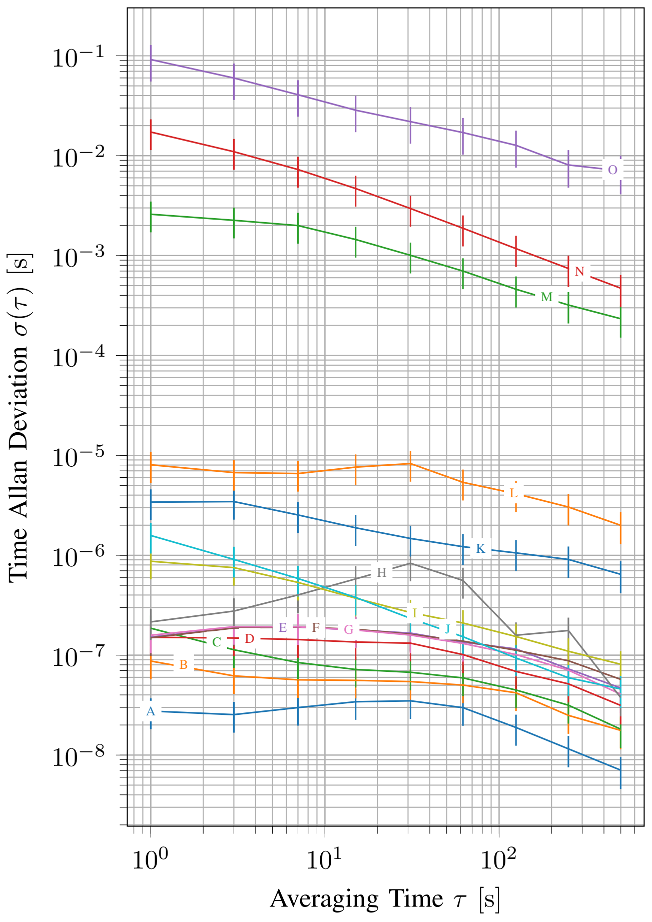

Fig. 9. Plot of the time Allan deviations of the measurements outlined in Table I. $\tau \in [1\,\mathrm{s}; 500\,\mathrm{s}]$.

of the PTP Grandmaster and slave devices, $\overline{\Delta p}$, in seconds. While this measurement is relevant to estimate the absolute synchronization precision of calibrated devices in the field, it is less relevant for this work: software-based adjustments can reliably compensate this measured phase offset during device calibration, if these offsets remain constant in the short and long term.

As a baseline measurement, the DUT is configured to use the hardware timestamp abilities of the DUT's Ethernet MAC. Furthermore, the software low-pass filter for captured PTP message timestamps is disabled. No additional system load through a process running on the Tock OS is simulated and syntonization through PTP is enabled. The device operates at 100 MHz and does not experience artificially generated interrupts. As can be seen in measurement series A, the DUT achieves a high degree of synchronization accuracy under these conditions: for $\tau = 1$, the time Allan deviation is at approximately 28 ns.

Notably, enabling the software low-pass filter on PTP *sync* and *delay measurement* messages when using hardware-acquired timestamps reduces the measured accuracy. This can be observed in measurement series E, compared to the

Measurement Series	Implemented Timestamp Point	Syntonization	Low-pass Filter	Userspace Workload	Random Interrupt Generator	Scheduling Algorithm	$\sigma_{\mathrm{tdev}}(\tau)$ [s], $\tau = 1, 7, 62$s	$\overline{\Delta p}$ [s]
A	Hardware (LiteEth MAC)	✓	✗	✗	✗	Cooperative	2.8e−8, 3.4e−8, 1.9e−8	−8.6e−7
B	Hardware (LiteEth MAC)	✓	✗	✗	✓	Cooperative	8.7e−8, 5.6e−8, 4.2e−8	−5.6e−7
C	Hardware (LiteEth MAC)	✓	✗	✓	✓	Priority (TAP network)	1.9e−7, 7.2e−8, 4.5e−8	−6.6e−7
D	LiteEth kernel driver	✓	✓	✗	✗	Cooperative	1.5e−7, 1.4e−7, 6.9e−8	−9.6e−6
E	Hardware (LiteEth MAC)	✓	✓	✗	✗	Cooperative	1.5e−7, 1.8e−7, 1.2e−7	−6.8e−7
F	Hardware (LiteEth MAC)	✓	✓	✓	✗	Cooperative	1.5e−7, 1.8e−7, 1.1e−7	−6.8e−7
G	Hardware (LiteEth MAC)	✓	✓	✗	✓	Cooperative	1.6e−7, 1.8e−7, 1.0e−7	−5.1e−7
H	Hardware (LiteEth MAC)	✗	✓	✗	✗	Cooperative	2.2e−7, 5.8e−7, 1.6e−7	−1.9e−6
I	LiteEth kernel driver	✓	✓	✓	✗	Round Robin	8.7e−7, 3.7e−7, 1.5e−7	−2.5e−5
J	LiteEth kernel driver	✓	✗	✗	✗	Cooperative	1.6e−6, 3.8e−7, 9.4e−8	−8.7e−6
K	Userspace (Packet callback)	✓	✓	✗	✗	Cooperative	3.4e−6, 1.9e−6, 1.1e−6	−6.4e−4
L	Userspace (Packet callback)	✓	✓	✓	✗	Priority (TAP network)	8.0e−6, 7.6e−6, 4.1e−6	−6.2e−4
M	Userspace (Packet callback)	✓	✓	✓	✓	Priority (TAP network)	2.6e−3, 1.4e−3, 4.6e−4	−6.9e−4
N	Userspace (Packet callback)	✓	✗	✓	✗	Round Robin	1.7e−2, 4.7e−3, 1.2e−3	−3.8e−3
O	Userspace (Packet callback)	✓	✗	✓	✗	Cooperative	9.2e−2, 2.9e−2, 1.3e−2	−2.4e−3

TABLE I
MEASUREMENT SERIES

otherwise identical measurement series A with the low-pass filter disabled. This behavior can be explained through the particular implementation strategy of the low-pass filter. The low-pass filter is implemented using a floating-window of the past n observations for *sync* and *delay measurements* respectively, and filtering the largest m_p positive and m_n negative deviations from the mean observed values. Even if all incoming measurements are precise, out of the last m observations there must be $2m$ observations with a higher deviation from the mean observed value for the current observation to not be filtered. This implies that, in contrast to measurement A, in measurement E the DUT's local PTP clock will occasionally be free-running for longer than $2^{\texttt{logSyncInterval}} = 1\,\mathrm{s}$ and thus experience clock noise and drift.

Disabling the PTP clock syntonization, and only performing a synchronization of the absolute PTP Grandmaster's time will also decrease the synchronization accuracy, as shown in measurement series H. Furthermore, the time Allan deviation increases with larger τ, before converging towards 0 along with the other plotted measurements. This can be potentially explained by the fact that both the PTP Grandmaster and DUT slave clocks experience frequency drift. As this drift is not corrected, it can become significant for some values of τ, whereas for $\tau \to \infty$ the continued absolute time synchronization makes the frequency offset between these clocks insignificant.

As shown by measurement series B, enabling the generation of randomly distributed interrupts through the random interrupt generator core, while acquiring PTP timestamps using the LiteEth MAC, does not significantly influence the observed results. Compared to measurement series A is has a marginally reduced accuracy. Measurement series E (without artificial interrupts) and G (with artificial interrupts) perform virtually identical. This confirms that the acquisition of hardware timestamps is – up to a certain degree – decoupled from other system aspects, specifically the CPU and OS behavior, such as handling of arriving interrupts.

However, as evident when comparing measurement series B and C, even when acquiring timestamps through the LiteEth MAC hardware of the FPGA, the synchronization accuracy can degrade under the presence of additional system load. A possible explanation for this observation is that this additional system load and incoming events introduce additional delay between the PTP captured timestamp point and the time at which the DUT's local PTP clock is adjusted. While the DUT's PTP implementation contains logic attempting to adjust and correct for this additional, system-internal delay, these observations hint at the fact that these compensation mechanisms are in fact not able to entirely compensate system-internal delays. Nonetheless, measurement series E and F show that these effects can in some cases be compensated by the PTP message low-pass filter.

When the PTP message timestamp is acquired in the LiteEth MAC kernel driver instead, at least under some conditions, similar performance compared to hardware-assisted timestamping can be achieved (refer to measurement series D). This result can be attributed to a number of observations: first of all, Tock has a conceptually simple method of reacting to interrupts and dispatching the corresponding interrupt handler. As an interrupt arrives, a context switch to the Tock kernel is performed if the system is currently running a process, halting any userspace process. Dispatching to a driver's interrupt handler can thus only be delayed by either an ongoing kernel operation (given that the kernel is cooperatively scheduled) or having pending interrupts with a higher priority. Thus, if neither of the two aforementioned conditions are present, the code-path from an arriving CPU interrupt to the kernel driver's interrupt handler is expected to have relatively deterministic timing characteristics. Because conditions producing additional delay when handling LiteEth MAC interrupts do occur in practice, for example because of higher-priority UART interrupts, using a low-pass filter significantly increases the synchronization accuracy here. This behavior, as observed with measurement series D and J, is in contrast to what can be seen with hardware-assisted timestamping in A and E. It thus further confirms the hypothesis postulated previously and shows that the low-pass filter on PTP message timestamps can be an effective means to improve on synchronization accuracy.

Moving the captured PTP message timestamp point into the LwIP userspace process itself, immediately after it has been scheduled because of an incoming packet or completed transmission of an outgoing packet by the kernel, further decreases the PTP synchronization accuracy. This is expected, as significantly more code is involved between the LiteEth MAC CPU interrupt, and the userspace application being scheduled because of the incoming packet. In addition to all delay and jitter sources when capturing the PTP message timestamp in the LiteEth MAC driver, in case of an incoming packet, the packet's contents have to be copied into a userspace buffer. Furthermore, a call to a function in userspace has to be scheduled by the kernel. Tock organizes calls into userspace applications in a queue structure. Thus, depending on the current system state and scheduling algorithm used, a function invocation in userspace has to wait for other pending function calls, other processes, as well as the application signaling to the kernel that it is able to receive a function call (called *yielding*).

When only a single process is running on the Tock OS, the scheduling algorithm does not meaningfully influence the system behavior: the OS will simply always schedule the single process as long as it is ready to execute. In contrast, the scheduling algorithm has a significant influence on the system behavior when running more than one process. Tock provides different scheduling algorithms, such as *Round Robin* which sequentially executes different processes in a loop, each with a given time-slice. The *Priority* scheduler always executes the highest-priority process which is ready to run, until it voluntarily gives up control. Furthermore, the *Cooperative* scheduler executes processes in a round-robin fashion until it voluntarily gives up control.

In theory this means that running a process assigned the highest priority with the priority scheduler implies this process has the same responsiveness and time scheduled as running it as the only process on the OS. As shown by measurement series K and L, this assumption does not hold in practice. This discrepancy between the expected and observed system behavior is suspected to be caused by two implementation artifacts of Tock: when a hardware interrupt or software event occurs and a userspace process has been running prior to the context switch into the kernel, the kernel must save the CPU state and other information about the previous application prior to calling the driver interrupt handler, taking additional time. Furthermore, the previous process could have started kernel operations which have observable effects even after the process has relinquished control, such as UART I/O operations.

Enabling generation of randomly-distributed interrupts has only limited effects on synchronization accuracy with the implemented PTP message timestamp point in the LiteEth MAC hardware or its kernel driver. In contrast to that, measurement series L and M show that the PTP accuracy significantly decreases by multiple orders of magnitude when the device is subject to frequent interrupts and PTP message timestamps are acquired in userspace. Because pending interrupts take priority over outstanding tasks of processes, the Tock scheduler will not schedule a userspace process until there are no more pending interrupts. Even when a process is running, upon an arriving interrupt, the system performs a context switch to kernel mode and handles any outstanding interrupts before the process is scheduled again. This implies a random and potentially significant delay prior to the userspace application being able to acquire a PTP message timestamp. Thus, the PTP synchronization accuracy decreases accordingly, in this case into the multi-millisecond range.

Even worse synchronization accuracy can be seen in measurement series N and O. These use the Round Robin and Cooperative scheduler, respectively. Thus the process running the LwIP network stack and acquiring the PTP message timestamps no longer has priority over the competing system workloads. This causes the acquired PTP message timestamps to have a very high delay and jitter, compared to the normative PTP message timestamp point as defined by IEEE 1588.

C. Interpretation of measurement results

The measurement results as presented in the previous section show how different implemented PTP message timestamp points and system characteristics influence the achievable PTP synchronization accuracy. Furthermore, the results allow to draw conclusions of whether a PTP implementation without hardware assistance is viable for specific application areas and highlight important system characteristics which can serve as design considerations for building systems.

In particular, an implemented PTP timestamp point close to the underlying transport hardware provides less opportunities for introducing delay and jitter between the implemented and reference PTP timestamp points. Keeping these values low is vital to achieve precise PTP clock synchronization and syntonization.

In the context of a microcontroller platform without hardware assistance for PTP, this implies that the PTP message timestamps should be acquired as closed to the underlying transport's interrupt handler as possible. However, care must be taken to reliably associated correct timestamps to incoming PTP messages: if using an embedded operating system such as Tock, which comprises a top-half and bottom-half interrupt handling concept, acquiring the PTP message timestamp in the top-half interrupt handler might not be possible. At this stage of the interrupt handling process, only the fact that an interrupt at the transport layer has occurred can be observed. A captured timestamp cannot necessarily be reliably associated with a particular packet as, for instance, multiple packets could have been received until the bottom-half interrupt handler is executed. Thus, the bottom-half interrupt handler of a transport layer is a viable target to acquiring PTP message timestamps which can be reliably associated with a particular PTP message.

To suppress statistical outliers in acquired PTP message timestamps, a low-pass filter on PTP messages as presented in this work can be an effective utility. However, it can also negatively influence the achievable PTP synchronization accuracy if the local PTP clock's instabilities outweigh potential errors in the acquisition of PTP message timestamps.

If implementing the PTP message timestamp point in the transport driver's interrupt handler or comparable low-level OS layers is not possible, implementing the PTP message timestamp point in upper OS layers up to and including userspace applications can be possible while retaining PTP synchronization accuracy in the multi-microsecond range. However, this makes the captured PTP message timestamps significantly more susceptible to other system behavior. System load and external events such as interrupts can significantly degrade PTP synchronization performance. Using priority or real-time scheduling algorithms can help to compensate for these effects.

The observed 1-PPS two-sample deviations for hardware-based timestamping of PTP messages are in the expected range of sub-microsecond precision. While generally results obtained when not utilizing hardware-based timestamping capabilities are less accuracy, they are still promising:

Lédeczi, Nádas, Völgyesi, *et al.* present a distributed system to accurately locate shooters in urban environments. Using their synchronization method, multi-microsecond synchronization accuracy can be achieved along their communication paths [21]. Furthermore, phase-angle synchronization in 60 Hz power grids requires multi-microsecond time synchronization accuracy [22]. Given specific system constraints and acquiring PTP message timestamps close to the interrupt handler, the DUT as presented can achieve multi-microsecond and even sub-microsecond synchronization accuracy; as such it appears viable that such use cases are implemented using commodity embedded systems not featuring hardware-based timestamping capabilities.

VIII. Conclusion

This paper investigated the feasibility of PTP-based time synchronization implementations in the context of embedded systems, without explicit PTP hardware support. In particular, it focused on determining estimates of the achievable accuracy of time synchronization and syntonization when utilizing IEEE 1588 PTP, using a commodity Ethernet-based network infrastructure and a PTP slave clock device without hardware-timestamping capabilities for PTP messages.

To answer the aforementioned question, first, the relevant details and mechanisms employed by the IEEE 1588 Precision Time Protocol standard have been reiterated. Following that, important system characteristics and constraints were analyzed. This served as a basis for designing a measurement system which is a reasonably close approximation of an embedded system requiring time synchronization. Based on the identified relevant characteristics, a measurement system architecture has been designed and implemented. This architecture specifies both, technical details of the hardware and software components involved in the measurement system, as well as methods to acquire and subsequently analyze the measured results. To process the acquired data sets and confirm basic assumptions of the previous sections, appropriate statistical methods and measures were used. Finally, the acquired measurement series have been presented, showing how the accuracy of time synchronization using PTP changes under different simulated internal and external system conditions. This information has been used to deduce a cause-effect relationship between the system conditions and the observed results, and subsequently to derive advice for designing such systems.

This paper shows that hardware-based timestamping of PTP messages is not necessarily a requirement for operating the IEEE 1588 Precision Time Protocol, at least for some application areas. Even under difficult system conditions with simulated workloads and external interrupts, a system using purely software-based PTP message timestamping can achieve a sub-microsecond two-sample deviation compared to the respective PTP Grandmaster clock. However, when acquiring PTP message timestamps in upper system layers such as userspace applications, the achievable time synchronization accuracy may decrease significantly. Thus, this should only be done when the embedded system's kernel or interrupt handling logic cannot be modified to acquire timestamps of PTP messages.

In this work, the performance of a single test system (Device Under Test) has been observed under different simulated system conditions. While care has been taken to design a measurement system and DUT which is representative for a broad range of microcontroller-based embedded systems, future research may determine the generalizability of these results. For example, different CPU architectures or multi-core systems may potentially show different behavior.

Furthermore, the simulated workloads used in this analysis may not be entirely representative of a real production workload. Analyzing the DUT's behavior while integrated in a production system may allow for further application area specific optimizations and can help to develop more advanced software techniques to compensate errors in the acquired PTP message timestamps.

References

[1] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std. 1588-2019*, 2020. DOI: 10.1109/IEEESTD.2020.9120376.

[2] S. Waldhauser, B. Jaeger, and M. Helm, "Time Synchronization in Time-Sensitive Networking," Apr. 2020. DOI: 10.2313/NET-2020-04-1_10.

[3] *PTP4l(8) System Manager's Manual*, 3.0, Man page, linuxptp, Apr. 2018.

[4] *Junos® OS Time Management Administration Guide*, Juniper Networks, Inc., Sunnyvale, CA, Apr. 2021. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/time-mgmt/time-mgmt.pdf (visited on 05/15/2021).

[5] H. Zhou, C. Nicholls, T. Kunz, and H. Schwartz, "Frequency Accuracy & Stability Dependencies of Crystal Oscillators," Nov. 2008. [Online]. Available: http://kunz-pc.sce.carleton.ca/Thesis/CrystalOscillators.pdf (visited on 07/15/2021).

[6] "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20190608-Priv-MSU-Ratified," RISC-V Foundation, Tech. Rep., Jun. 2019. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMFDQC-and-Priv-v1.11/riscv-privileged-20190608.pdf (visited on 07/15/2021).

[7] K. C. Wang, *Embedded and Real-Time Operating Systems*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-51517-5_8.

[8] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices," Jan. 2018. DOI: 10.14722/ndss.2018.23176.

[9] M. D. Marieska, P. G. Hariyanto, M. F. Fauzan, A. I. Kistijantoro, and A. Manaf, "On performance of kernel based and embedded Real-Time Operating System: Benchmarking and analysis," in *2011 International Conference on Advanced Computer Science and Information Systems*, 2011, pp. 401–406, ISBN: 978-979-1421-11-9.

[10] *NetFPGA-1G-CML Reference Manual*, Digilent Inc. [Online]. Available: https://reference.digilentinc.com/programmable-logic/netfpga-1g-cml/reference-manual (visited on 06/04/2021).

[11] F. Kermarrec, S. Bourdeauducq, J. L. Lann, and H. Badier, "LiteX: An open-source SoC builder and library based on Migen Python DSL," 2020. arXiv: 2005.02506. [Online]. Available: https://arxiv.org/abs/2005.02506 (visited on 07/15/2021).

[12] H. Xu, Z. Chen, M. Sun, and Y. Zhou, "Memory-Safety Challenge Considered Solved? An Empirical Study with All Rust CVEs," vol. abs/2003.03296, 2020. arXiv: 2003.03296. [Online]. Available: https://arxiv.org/abs/2003.03296 (visited on 07/15/2021).

[13] A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis, "Multiprogramming a 64kB Computer Safely and Efficiently," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP'17, Shanghai, China: ACM, Oct. 2017, pp. 234–251. DOI: 10.1145/3132747.3132786.

[14] A. Levy, B. Campbell, B. Ghena, P. Pannuto, P. Dutta, and P. Levis, "The Case for Writing a Kernel in Rust," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, ser. APSys '17, Mumbai, India: ACM, Sep. 2017, 1:1–1:7. DOI: 10.1145/3124680.3124717.

[15] *Time Tagger Series*, Product Brochure, Swabian Instruments GmbH, Stuttgart, Germany. [Online]. Available: https://www.swabianinstruments.com/static/downloads/TimeTaggerSeries.pdf (visited on 06/14/2021).

[16] W. Riley and D. Howe, *Handbook of Frequency Stability Analysis*, Special Publication (NIST SP) - 1065, Gaithersburg, MD, Jul. 2008. [Online]. Available: https://www.nist.gov/publications/handbook-frequency-stability-analysis (visited on 06/15/2021).

[17] J. E. Gray and D. W. Allan, "A Method for Estimating the Frequency Stability of an Individual Oscillator," in *28th Annual Symposium on Frequency Control*, Atlantic City, NJ, Apr. 1974, pp. 243–246. DOI: 10.1109/FREQ.1974.200027.

[18] F. Vernotte, M. Addouche, J. Delporte, and M. Brunet, "The three cornered hat method: an attempt to identify some clock correlations," in *Proceedings of the 2004 IEEE International Frequency Control Symposium and Exposition, 2004.*, Montreal, QC, Canada, Aug. 2004, pp. 482–488. DOI: 10.1109/FREQ.2004.1418506.

[19] A. Premoli and P. Tavella, "A revisited three-cornered hat method for estimating frequency standard instability," *IEEE Transactions on Instrumentation and Measurement*, vol. 42, no. 1, pp. 7–13, 1993. DOI: 10.1109/19.206671.

[20] W. J. Riley. "The Basics of Frequency Stability Analysis." (Aug. 2004), [Online]. Available: http://www.wriley.com/paper2ht.htm (visited on 07/07/2021).

[21] Á. Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, and G. Simon, "Countersniper System for Urban Warfare," *ACM Trans. Sen. Netw.*, vol. 1, no. 2, pp. 153–177, Nov. 2005, ISSN: 1550-4859. DOI: 10.1145/1105688.1105689.

[22] *Precise Timing for Power Industries and the Smart Grid*, Meinberg Funkuhren GmbH & Co KG. [Online]. Available: https://www.meinbergglobal.com/english/industries/smart-grid-timing.htm (visited on 07/14/2021).